Towards Developer- and task-tailored navigation models

Kevic, Katja ; Fritz, Thomas

Towards Developer- and Task-Tailored Navigation Models

Katja Kevic, Thomas Fritz
University of Zurich
Department of Informatics
Zurich, Switzerland
{kevic,fritz}@ifi.uzh.ch

ABSTRACT

Developers spend a substantial amount of their time navigating source code to locate the places which are relevant for the task at hand. Current recommendation approaches that support a developer in this costly activity, accentuate different aspects of code structure or a developer's interaction to infer the elements a developer might want to navigate to next. While some recommendation approaches use models that, for instance, focus predominantly on the source code topology, such as method call relations, others focus on the frequency and recency of previously visited code elements. All of these approaches generally use the same underlying model throughout the change task session, regardless of the specific developer or the type of change task. Thus, they have the implicit premise that developers' information needs and navigation behaviour stays the same over time and across developers and tasks. In on our research on developers' navigation, we saw, however, that a developer's navigation behaviour varies a lot over the course of a change task and across developers. This suggests that a model that dynamically adapts to the developer's navigation behaviour and information needs could provide much better support in recommending relevant elements to developers. The goal of the presented work is to investigate how a developer's information needs and navigation behaviour change over the course of a change task. With a better understanding, we will be able to develop recommendation models that better tailor the recommendations to the developer and the tasks and thus provide more relevant recommendations.

Categories and Subject Descriptors

D.2.6 [**Software Engineering**]: Programming Environments

General Terms

Human Factors

Keywords

Navigation models, developer, change task

CSD '14 Hong Kong

1. INTRODUCTION

When a developer gets a change task assigned, she spends a considerable amount of her time navigating the source code to locate the relevant places for solving the change task [6]. Since a developer unfamiliar with the system has to rely on her intuition to decide which elements to further visit, different approaches have been developed to proactively point developers to elements worth visiting next [15]. These navigation recommendations are based on different models of developer's information needs and navigation, such as the recency and frequency of previously visited elements [13], the source code topology [15, 7], the change history of the project [17, 18], or the textual similarity to the change task [9]. While each model emphasizes a specific and relevant aspect of the developer's navigation behaviour, all these approaches have in common that they look at a navigation session as an unvarying process, premising that a developer's questions and information needs stay the same for each change task and during the whole navigation session. A recent head-to-head empirical evaluation of eight navigation recommendation approaches compared each approach's recommendations to the navigation steps actually taken by study participants [14]. Within the top ten results, the best two approaches hit near or slightly above 50% of the actual study participants navigations. We hypothesize that the approaches evaluated in this comparison achieved such a relatively low accuracy because the information and navigation needs which arise during a navigation session change over time. We further hypothesize that using a single static model for identifying recommendations cannot satisfy the various needs that arise sometimes simultaneously. In previous studies, we already observed that developers' navigations vary across tasks and developers when performing change task [3].

The goal of this research is to investigate how a developer's information needs change over the course of a navigation session, as well as across developers and tasks. As a first step towards this goal, we focus on investigating how the frequency and recency of a developer's revisits of code elements change over the course of a change task. This first step is inspired by the finding of Piorkowski et al. who demonstrated in their empirical evaluation that a navigation recommendation model based on the recency of previously visited elements is between the best in terms of accuracy [14].

Our preliminary findings of an exploratory analysis of 49 developer navigation histories from a small open source project suggest that the repeated visiting of an element varies regarding a temporal aspect with the type of change tasks. Furthermore, our preliminary findings suggest that

navigation support based on the recency and frequency of previously visited elements should adapt to each developer's individual readiness to explore new elements during the course of navigation. Overall, these results indicate that models that take into account characteristics of the developer and the task might allow us to better model the navigation of developers and thus provide better support in the future.

2. RELATED WORK

For a variety of software engineering activities, such as fixing a bug, implementing a new feature or documenting code, developers establish a context of code elements relevant to the task [8, 3]. There are several research approaches that help to identify relevant code elements and relations between these elements. The approaches most related to our work proactively recommend where to navigate next once the developer is exploring the code base, such that parts of the particular context can be quickly identified. These approaches mainly differ in their model of developers' information needs they use to come up with recommendations, ranging from models based on source code topology [15, 7], textual features of the change tasks and the source code [9], combinations of these [10, 4], change history of a project [17, 18], all the way to interaction histories [2, 13, 11]. Most similar to our work are the last mentioned models that are based on a developer's interactions with source code elements. These models can further be divided into the models that rely on a sufficiently large history of interactions for a project [2, 11] and the model proposed by Parnin and Gorg that is based on the frequency of very recently visited elements [13]. All of these approaches however use fairly general models to recommend the next steps to navigate to and do not capture differences of developers and tasks or the differences while performing a change task.

3. EXPLORATORY ANALYSIS

In this exploratory analysis we investigate the revisiting of source code elements by a developer during a change task as a first step towards a better understanding and a better model of the navigation behaviour of developers. Revisiting code elements is a frequent activity during a navigation session, as the findings of an observational study show [3] and might be pivotal to the understanding of code during change tasks. In this study, we focus on the following three research questions:

RQ1 Does the ratio between navigation steps and revisitations differ across developers and tasks?

RQ2 Does the ratio between navigation steps and revisitations differ within tasks?

RQ3 Is the revisitation pattern within a task developer-specific?

3.1 Data set

To answer these research questions we analyzed interaction histories of the open source project Mylyn.Context (37 kLOC). We chose Mylyn.Context because it offers a reasonable amount of change tasks with interaction histories attached, and there is a reasonable amount of different developers that submitted their interaction histories for this project. The interaction histories for this project were captured as task contexts by the Eclipse plug-in Mylyn [5]. Mylyn captures for each change task the source code elements which were selected and edited by the developer and persists them as so called interaction events along with a timestamp in an XML file. One change task can have multiple interaction histories attached. The change tasks along with the attached interaction histories can be downloaded from the change task repository Bugzilla [1].

We filtered all available interaction histories of this open source project based on two criteria: 1) the interaction history includes at least seven navigation steps, and 2) the author of the interaction history has in sum at least five interaction histories submitted for this project. We used seven navigation steps as the minimum size for this analysis, as people are able to memorize an average of 7±2 items [12] and hence analyzing revisits for less than 7 navigation steps that could all be memorized seemed not valuable. Further we analyzed interaction histories for developers which have at least five interaction histories submitted for the project, as we aimed to investigate whether differences between developers exist and also whether attributes of interaction histories of the same developer differ. Our filtering scheme resulted in a total of 49 interaction histories belonging to 35 different change tasks. These interaction histories were submitted by four distinct authors. All of these four developers also contributed to Mylyn projects other than Mylyn.Context. Overall projects, developer D1 is the most active and worked on more change tasks than D2, D3 and D4. Within Mylyn.Context, developers D1 and D2 submitted substantially more interaction histories than the other two developers (see Table 1). Table 1 also depicts information regarding the average number of navigation steps present in each interaction history (i.e. the context size) and the average number of different elements present in each interaction history. A Mann-Whitney test indicates that the process of establishing a context, i.e. the number of navigation steps performed, varies significantly between the four developers. Developers D1 and D2 performed significantly less navigation steps than developer D3 ($p = .004$, respectively $p = .025$). However, the relatively high standard deviation associated with each developer's average context size indicates that the context size is further influenced by other variables. The developers also differ in the amounts of unique elements present in their interaction histories. A Mann-Whitney test indicates that developers D1, D2 and D4 look at significantly less unique elements within a navigation session than developer D3 ($p = .003$, $p = .003$, and $p = .042$). This finding is consistent with previous research [16, 3].

Since we observed that the standard deviation is particularly high for each developer's average context size and amount of different elements included in the context, we further investigated whether change task types influence these two variables. Table 1 presents the average context size and the average number of unique elements within a contexts grouped by the severity types of the change tasks. Our data set includes change tasks assigned to six different severity types. The largest group of change tasks was classified as enhancement, whereas only one change task was classified as critical. A Mann-Whitney test shows that the context size of an enhancement is significantly larger than the context size of a trivial change task ($p = .031$), of a normal change task ($p = .031$), and of a major change task ($p = .005$). In addition, a Mann-Whitney test shows that developers visited significantly more unique elements when working on an enhancement than when working on a trivial change task

($p = .037$), on a normal change task ($p = .041$), or on a major change task ($p = .019$).

3.2 Procedure and Results

We applied two preprocessing steps to the interaction histories of our data set. First, we filtered out all interaction events that are not associated with a selection or an edit. Second, after sorting the remaining interaction events in a chronological order, we filtered out the interaction events which ultimately followed an interaction event pointing to the same source code element. This way, we consider the selection of a particular source code element and the possible subsequent edit of the same source code element only once and do not count them as a revisitation.

RQ1: Does the ratio between navigation steps and revisitations differ across developers and tasks? To answer this research question we analyzed for each developer how many elements were visited more than once within an interaction history of a change task and how many navigation steps lie between the repeated visit of an element. Then, we calculated the ratio between the elements visited repeatedly and the total number of different elements within an interaction history. Table 1 summarizes for each developer the average ratio of revisited elements and the average number of navigation steps performed before revisiting an element.

Our results support previous findings [3], further showing that individual developers revisit considerably different amounts of source code elements. A Mann-Whitney test showed that the ratio of revisited elements of developer D1 and D2 were significantly higher than the ratio of revisited elements of developer D4 ($p = .035$, respectively $p = .025$). Furthermore, the number of navigation steps between the repeated visit of an element, differs considerably between individual developers as well. A Mann-Whitney test showed that developer D1, D2 and D4 revisits elements after significantly less navigation steps than developer D3 ($p = .003$, $p = .034$, respectively $p = .032$). These observation suggest that the amount of elements, which are revisited, and the steps performed between a revisitation are developer-specific.

Table 1 shows the average ratio of revisited elements and the average number of steps between the revisitations categorized according to different severity types appearing in our data set. The average revisitation ratios do not differ significantly between the different change task types. However, we observed a significant difference of the average number of steps developers performed between revisitations. A Mann-Whitney test showed that developers performed significantly more navigation steps between revisiting an element when working on an enhancement than when working on a major change task ($p = .008$) or on a trivial task ($p = .025$). Although we only analyzed a small data set, these observations suggest that developers revisit a lot of elements regardless on the specific type of change task, but that the revisitation behavior within a change task may vary.

RQ2: Does the ratio between navigation steps and revisitations differ within tasks? To find out whether the analyzed interaction histories include any aggregation of revisits, we grouped the revisited elements in so called revisit sessions. We defined that a revisit session is established if a developer revisits an element more than once within at most seven navigation steps. The revisit session is then extended to every further revisited element not more than seven navigation steps away. Note that one revisit session captures the revisits of only one unique source code element. In this analysis we only considered revisits to *methods*. We excluded two navigation histories of developer D1, one navigation history of developer D2 and two interaction histories of developer D4 that did not contain any revisited methods.

Once we found all revisit sessions for each interaction history, we plotted the appearance of a revisit session along the time-line of the interaction history. The time-line reflects all navigation steps performed by the developer within the navigation session. As an example, Figure 1 depicts the plot of the interaction history attached to change task 335606, which includes 83 navigation steps. The bubbles, each one reflecting a revisit session for a particular method, are located on the time-line according to their start. The size of the bubble reflects the intensity of the revisit session, i.e. the bigger the bubble the more times the specific method was visited within the particular revisit session.

We found two specific reoccurring patterns of revisit aggregation in all plotted revisit sessions. The first pattern includes an increased aggregation of revisit sessions in the very beginning of the change task and no further revisit session towards the ending of the navigation session. In our 44 plotted navigation sessions we found six occurrences of this pattern. Figure 1 depicts an example of a navigation session with revisit sessions matching this pattern. Further investigation into the descriptions of the change tasks matching the pattern revealed that all these change tasks have in common that they point to a location in the code either through a stack trace or through carefully describing a misbehavior of a specific UI element. Thus, we hypothesize that these change tasks do not require an exploratory phase by the developers. In this case, most beneficial would be a model that initially focuses on recency or frequency and favours the revisiting of elements, while later on the model should favour the exploration of new elements.

The second pattern we identified includes an increased aggregation of revisit sessions towards the end of the navigation session, while in the beginning of the navigation session no major revisit sessions are present. Seven out of 44 interaction histories analyzed match this pattern. A plot of a navigation session which revisit sessions match this second pattern can be found in Figure 2. Analyzing the descriptions and discussions of the according change tasks revealed that it was unclear how to reproduce the misbehavior and what the best design for performing the change task might be. Thus, we hypothesize that navigation sessions with revisit sessions that match the second identified pattern include an initial exploratory phase of the source code. In this case, most beneficial would be a model that initially focuses on exploring new elements, while later on the model should focus on recency or frequency and favour the revisitation of elements. Further analysis is needed to assign the remaining 31 interaction histories to a category.

RQ3: Is the revisitation pattern within a task developer-specific?
To analyze our last research question we identified the number of navigation steps each developer performed until no more new source code elements were added to the interaction history. We analyzed the data at class and at method

Table 1: Summary of the data set used in our exploratory analysis. The upper part of the table depicts the data set ordered by developer and the bottom half depicts the data set ordered by task type. In total, our data set includes 49 interaction histories with six different task types, submitted by four developers.

	# contexts	avg context size (SD)	avg unique elements (SD)	avg ratio of revisitation (SD)	avg steps between revisitations (SD)
Developer					
D1	19	85.6 (99.3)	44.6 (58.1)	0.419 (0.208)	10.60 (8.53)
D2	18	103.5 (97.2)	45.3 (33.9)	0.462 (0.205)	16.44 (11.31)
D3	7	264 (266.3)	124 (63.8)	0.310 (0.153)	29.14 (13.14)
D4	5	103.6 (72.1)	50.8 (26.0)	0.220 (0.134)	16.62 (9.14)
Task type					
critical	1	19(-)	8 (-)	0.875 (-)	3.75 (-)
trivial	3	31.7 (31.2)	17 (16.8)	0.499 (0.138)	5.15 (3.13)
normal	6	62.7 (52.2)	31.7 (22.7)	0.459 (0.280)	12.26 (7.09)
minor	4	75.3 (18.8)	32.8 (17.0)	0.391 (0.245)	10.63 (1.71)
enhancement	24	183.2 (179.0)	85.3 (65.4)	0.386 (0.201)	22.53 (13.19)
major	11	60.9 (30.0)	32.4 (15.9)	0.327 (0.113)	9.85 (4.50)

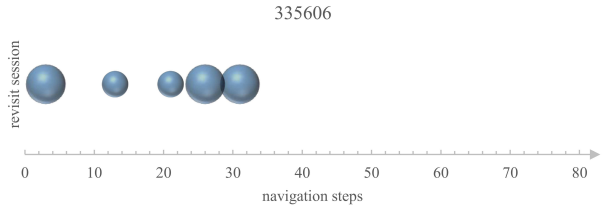

Figure 1: Revisit session plotted for change task 335606. The revisit sessions of this navigation show an increased aggregation of revisit sessions in the beginning.

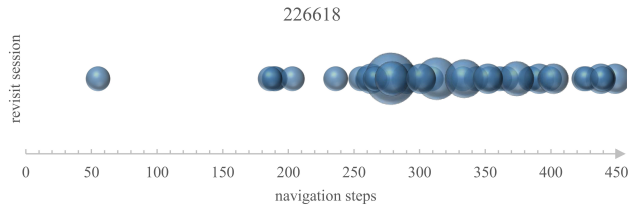

Figure 2: Revisit session plotted for change task 226618. This example has an increased aggregation of revisit sessions towards the end.

level and report the average percentage of navigation steps which were performed by each developer until no more new elements were visited. Developer D1 looked at new methods within the first 43% ($SD = 21\%$) of all navigation steps and within the first 24% ($SD = 12\%$) of all navigation steps new classes were explored. Developer D2 looked at new methods within the first 38% ($SD = 13\%$) and at new classes within the first 36% ($SD = 11\%$) of all navigation steps, developer D3 looked at new methods within the first 44% ($SD = 4\%$) and at new classes within the first 39% ($SD = 6\%$) of all navigation steps, and developer D4 looked at new methods within the first 37% ($SD = 6\%$) and at new classes within the first 27% ($SD = 14\%$) of all navigation steps.

Note that these numbers are averaged over all navigation histories, also including the navigation histories which match the first pattern identified in attempting to answer RQ2. These findings suggest that using a navigation recommendation tool which encourages developers to explore new source code elements, even after a short amount of navigation, does not satisfy the information needs of the developers. A navigation recommendation approach based on recency and frequency of previously visited elements seems to be more fitting. Furthermore, the individual developers seem to have a different readiness to explore new elements during their navigation session. For example, developer D1 and D2 looked on average across their interaction histories at about an equal number of unique source code elements, while developer D1 stopped exploring new classes after 24% of all navigation steps and developer D2 stopped after 36% of all navigation steps. This suggest that the potential switch between navigation recommendation approaches is developer-specific.

3.3 Threats

The external validity of our exploratory analysis is threatened by the small number of analyzed interaction histories captured by only a few developers, belonging only to one specific project. Also, the expertise and experience of each developer might vary a lot and influence their behavior of navigating throughout the source code. Furthermore, interaction histories captured through Mylyn might bare noise, as developers have to manually start and stop the recording of the interactions within the IDE. We are aware of these threats. Future studies are planned to investigate our results which can be generalized more.

4. DISCUSSION

Inspired by the findings of our preliminary analysis we plan to investigate a navigation recommendation model, which is tailored to the developer, to the current change task and to varying questions a developer may have in mind while navigating source code. For tailoring the model to the developer, we plan to analyze previous interaction histories of developers to predict a developer-specific "intensity" of a recommendation model. As an example, in our analysis we observed that developer D1 would profit more than developer D4 of

more frequent recommendations based on recency. Similarly, we want to investigate whether other models, such as models which are based on the lexical cues within the source code, exhibit similar developer-specific intensities. At this stage of our research, tailoring the navigation model to tasks is mainly engaged with estimating the appropriate context size. While the estimation of an appropriate context size is important to not overload developers with recommendations, the appropriate context size might be also important when summarizing contexts for task resuming. Furthermore, we plan to tailor our navigation recommendation model also to cues within the task. As we observed in RQ2, developers are interested in visiting different types of elements at different stages of their navigation session. This change of interest could be addressed by varying the recommendation models within the task. We plan for future research to detect behaviors of developers which indicate that the developer is going to be interested in a recommendation stemming from a particular kind of source. The observations for RQ3 further suggest that these models should be developer-specific as well.

5. CONCLUSION

We investigated developers' revisitations of elements as a first step towards our main interest of finding out whether developers' information needs change over the course of a navigation session and how these needs can be addressed. The preliminary findings from our exploratory analysis indicate that individual developers revisit elements frequently, however, that the revisits vary significantly across developers and also vary significantly for a single developer over the course of a change task. Furthermore, our findings indicate that developers tend to follow specific patterns of revisitation according to the type of change task. These patterns suggest that a dynamically adapting model for navigation recommendation during the course of a navigation session may be beneficial.

6. REFERENCES

[1] https://bugs.eclipse.org/bugs/.

[2] R. DeLine, A. Khella, M. Czerwinski, and G. Robertson. Towards understanding programs through wear-based filtering. In *Proceedings of the 2005 ACM Symposium on Software Visualization*, SoftVis '05, pages 183–192, New York, NY, USA, 2005. ACM.

[3] T. Fritz, C. Sheperd, K. Kevic, W. Snipes, and C. Bräunlich. Developers' code context models for change tasks. In *Foundations of Software Engineering*, 2014.

[4] E. Hill, L. Pollock, and K. Vijay-Shanker. Exploring the neighborhood with dora to expedite software maintenance. In *Proceedings of the Twenty-second IEEE/ACM International Conference on Automated Software Engineering*, ASE '07, pages 14–23, New York, NY, USA, 2007. ACM.

[5] M. Kersten and G. C. Murphy. Using task context to improve programmer productivity. In *Proceedings of the 14th ACM SIGSOFT International Symposium on Foundations of Software Engineering*, SIGSOFT '06/FSE-14, pages 1–11, New York, NY, USA, 2006. ACM.

[6] A. Ko, B. Myers, M. Coblenz, and H. Aung. An exploratory study of how developers seek, relate, and collect relevant information during software maintenance tasks. *Software Engineering, IEEE Transactions on*, 32(12):971–987, Dec 2006.

[7] T. LaToza and B. Myers. Visualizing call graphs. In *Visual Languages and Human-Centric Computing (VL/HCC), 2011 IEEE Symposium on*, pages 117–124, Sept 2011.

[8] T. D. LaToza, G. Venolia, and R. DeLine. Maintaining mental models: A study of developer work habits. In *Proceedings of the 28th International Conference on Software Engineering*, ICSE '06, pages 492–501, New York, NY, USA, 2006. ACM.

[9] J. Lawrance, R. Bellamy, and M. Burnett. Scents in programs:does information foraging theory apply to program maintenance? In *Visual Languages and Human-Centric Computing, 2007. VL/HCC 2007. IEEE Symposium on*, pages 15–22, Sept 2007.

[10] J. Lawrance, M. Burnett, R. Bellamy, C. Bogart, and C. Swart. Reactive information foraging for evolving goals. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '10, pages 25–34, New York, NY, USA, 2010. ACM.

[11] S. Lee, S. Kang, and M. Staats. Navclus: A graphical recommender for assisting code exploration. In *Proceedings of the 2013 International Conference on Software Engineering*, ICSE '13, pages 1315–1318, Piscataway, NJ, USA, 2013. IEEE Press.

[12] G. A. Miller. The magical number seven, plus or minus two: Some limits on our capacity for processing information. *The Psychological Review*, 63(2):81–97, March 1956.

[13] C. Parnin and C. Gorg. Building usage contexts during program comprehension. In *Proceedings of the 14th IEEE International Conference on Program Comprehension*, ICPC '06, pages 13–22, Washington, DC, USA, 2006. IEEE Computer Society.

[14] D. Piorkowski, S. Fleming, C. Scaffidi, L. John, C. Bogart, B. John, M. Burnett, and R. Bellamy. Modeling programmer navigation: A head-to-head empirical evaluation of predictive models. In *Visual Languages and Human-Centric Computing (VL/HCC), 2011 IEEE Symposium on*, pages 109–116, Sept 2011.

[15] M. P. Robillard. Automatic generation of suggestions for program investigation. *SIGSOFT Softw. Eng. Notes*, 30(5):11–20, Sept. 2005.

[16] M. P. Robillard, D. Shepherd, E. Hill, K. Vijay-Shanker, and L. Pollock. An empirical study of the concept assignment problem. Technical Report SOCS-TR-2007.3, School of Computer Science, McGill University, 2007.

[17] A. Ying, G. Murphy, R. Ng, and M. Chu-Carroll. Predicting source code changes by mining change history. *Software Engineering, IEEE Transactions on*, 30(9):574–586, Sept 2004.

[18] T. Zimmermann, A. Zeller, P. Weissgerber, and S. Diehl. Mining version histories to guide software changes. *Software Engineering, IEEE Transactions on*, 31(6):429–445, June 2005.